SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 24, 2000	FINAL

4. TITLE AND SUBTITLE
Support for Developing an Object-Oriented Simulation/Modeling Environment to Enhance C3I Simulation/Modeling

5. FUNDING NUMBERS
DAAH04-93-G-0401

6. AUTHOR(S)
Albert A. Fredericks

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Monmouth University
West Long Branch, NJ 07764

8. PERFORMING ORGANIZATION REPORT NUMBER
32318-EL

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 32318.1-EL

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

20000707 072

13. ABSTRACT *(Maximum 200 words)*

The main objective of this research was to define, design and prototype an open, object-oriented environment consisting of a variety of integrated tools for modeling, analysis, simulation and engineering of computer / communications and other systems such as information systems, business operations and production systems. This environment was to overcome certain problems seen with simulation environments that existed at the time – and still persist today. These problems include the lack of: i) a simple to use yet effective performance modeling tool set and ii) an open, integrated, object-oriented performance modeling environment using a common programming language and promoting reuse. This report summarizes the results obtained in addressing these problems including an overview of the tools and environments that were prototyped as part of this effort as well as their applications to the performance analysis of some specific communications systems. Issues related to training and education in the use of these tools and environments are also addressed.

14. SUBJECT TERMS	15. NUMBER IF PAGES
Modeling and simulation tools; integrated modeling environments; distributed simulation; performance analysis.	36
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Final Report for ARO Grant Number: DAAH04-93-G-0401

(Title: Support for Developing an Object-Oriented Simulation / Modeling Environment to Enhance C^3I Simulation/Modeling)

Forward

It is well recognized that the use of appropriate tools and techniques for performance modeling, analysis and simulation can provide quantitative insight into system performance that would otherwise be difficult, too expensive or even impossible to obtain. The increased importance of such tools and techniques for ensuring cost-effective performance engineering of computer / communications systems has placed an increased premium on the ease of use of tools and on the reuse of models. While there have been many advances and improvements in simulation environments, there still seems to be two important capabilities that are missing: i) a simple to use yet effective performance modeling tool set and ii) an open, integrated, object-oriented performance modeling environment using a common programming language and promoting reuse. The former would allow systems engineers to quickly learn the important aspects of performance modeling and to do their own quick, high-level systems modeling when needed. The later would significantly increase the available pool of potentially contributable models as well contributors. It would also provide an excellent environment for education and training. The main purpose of this research was to address these two problem areas.

Table of Contents

Appendices

Illustrations

Tables

2. Research and Prototype Development Under This Grant

Performance modeling and analysis provide an important means for ensuring the cost effective design, development, engineering and operations of computer / communications systems as well as other systems such as manufacturing and information systems and for process reengineering. Appropriate use of performance modeling and analysis methods and tools can provide quantitative insight into system performance that would otherwise be difficult, too expensive or even impossible to obtain. This is especially true for very large, complex systems when performance under adverse conditions is extremely important. Both DOD organizations as well as industry have used and continue to use a diverse set of simulation tools as well as contractors to meet their growing simulation and modeling needs. Some simulation tools are specifically geared toward communications systems modeling, e.g., [1][†], [2], [3], [4] while others are more generic in nature, e.g., [5], [6], [7]. (A discussion of the role of these and other packages in modeling communications systems can be found in [8].) However, there are still many shortcomings and problems associated with the current methodologies and supporting environments for providing quantitative support for the design, development, engineering and operations of systems; shortcomings and problems that can lead to high costs and long lead times for model development, costly and time consuming (and often deficient) verification / validation procedures and great difficulties in interfacing various subsystem models to model "systems of systems" – a growing need for DOD systems as well as those of many industries. The purpose of this research effort was to address some of these shortcomings and needs. In addition to defining, designing and prototyping performance modeling tools and environments that provide some solutions, the important issue of education and training in the use of such tools and environments is also addressed.

2.1 Statement of Problem Studied

The main objective of this research was to define, design and prototype an open, object-oriented environment consisting of a variety of integrated tools for modeling, analysis, simulation and engineering of computer / communications and other systems such as

[†] Simple numbered references can be found in Section 4: References. Reference numbers preceded by PT can be found in Section 2.3: Publications and Technical Reports.

information systems, business operations and production systems. This environment was to overcome certain problems seen with simulation environments that existed at the time – and still persist today. These problems include the lack of: i) a simple to use yet effective performance modeling tool set and ii) an open, integrated, object-oriented performance modeling environment using a common programming language and promoting reuse. (Simulation model reusability issues have been a concern for some time, e.g., see [9].)

2.1.1 High-Level Modeling Tools

Often during a system's life cycle, particularly in the early stages of system definition and design, there is a need for quickly obtaining estimates of system performance measures. Moreover, even when detailed simulation models are needed, it is highly desirable to have the ability to quickly build independent, high-level simulation models for verification. In today's environment, there is a relatively rapid turnaround in employees; key personnel often leave positions in an organization either for other positions within the organization or to join other organizations. Therefore, it is highly desirable to have a generic, performance-modeling environment that can be quickly learned and, in fact, one that assists in the overall performance analysis education of new employees. One aspect of this research effort was to define, design and prototype an environment addressing these issues – including use in the education process.

2.1.2 Open Environment

At the time that this effort was initiated, there was a heavy reliance on detailed simulation models built with proprietary software embedded in closed simulation building environments. This often resulted in long delays and costly overruns in model building. Moreover, the reuse of models built in these environments was essentially nonexistent. Since then a concerted effort has been made to try to develop and implement interface standards that would enhance model interoperability and reuse. The High Level Architecture (HLA) definition has been a primary factor in moving toward this end. This certainly increases the value of using any currently available simulation modeling environments that are HLA compliant. However, there are many cases where writing simulation modules in an object-oriented high level language (e.g., C++) would be far more efficient than using an existing tool. For example, some of the most used simulation modeling tools require a considerable time to learn; by contrast, there is a very large number of proficient C++ programmers. However, for an open, simulation modeling, environment to be a valuable asset, it must have an adequate set of desirable features – and be able to meet HLA requirements. Another objective of this research was to define, design and prototype an open simulation modeling environment promoting reuse and with a rich set of features that would make it a valuable asset in any collection of simulation modeling tools. Such desirable features include: i) an integrated toolkit supporting various aspects of performance modeling, analysis and engineering; ii) an open system view supporting the addition and integration of externally developed modules; iii) heterogeneous module support allowing integration of disparate modules; iv) hierarchical modeling support, including support for high-level analytic models; v) support for distributed simulation and, vi) programmable nodes. In addition, to further promote reuse, there should be a minimal set of requirements for modules that are to be

added to the environment. (Note that while HLA compliance ensures that a model can be reused, it does not guarantee that this will be easy to do!)

2.1.3 One of Several Tools

It is important to note that the environment and tools discussed here are not meant to be sole replacements for all existing environments and tools. Indeed, most of the modeling and simulation tools that exist today can be and are extremely useful. The problem as we see it is that any major systems engineering and development organization needs a variety of tools to meet its needs. Figure 1 illustrates this in a qualitative way – and points out some of the inherent problems of misuse. Generally, in order to model more complex systems, we need more complex tools, i.e., tools that are more difficult to use, costly to purchase, etc. The curve (straight line) on the figure indicates a hypothetical "ideal" tradeoff for these quantities. We can conceptualize various tools on this figure, hopefully lying along or below this tradeoff curve - e.g., Q+ and OPNET as indicated for illustrative purposes. (Q+ and OPNET, while on opposite ends of the curve, are probably two of the better simulation modeling tools – within their region of applicability.) However, if these tools are misused, i.e., used to model systems outside their range of applicability, one may find that an enormous price in complexity of use might be paid. For example, while some very complex systems can be analyzed with Q+, its use in some such cases may be far more complex and time consuming than writing code from scratch. On the other hand, using OPNET to model relatively simple systems - or those that don't readily fit in OPNET's finite state machine paradigm - may also introduce unnecessary complexity.

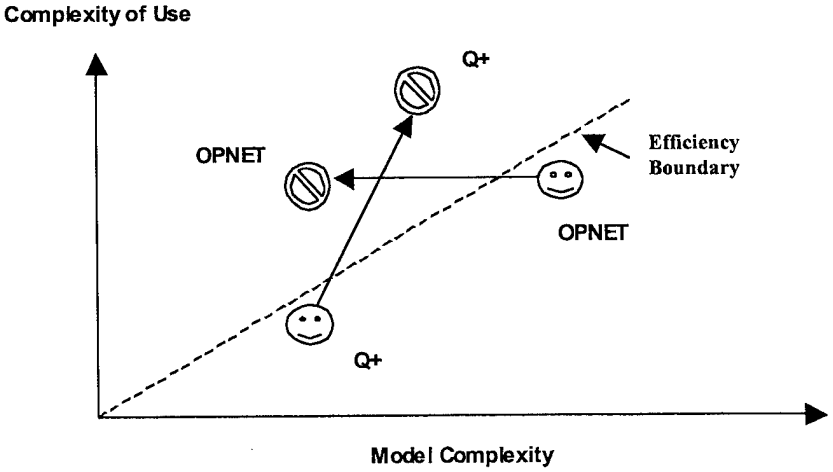


Figure 1 - Complexity of Tool Use vs. Complexity of Model Needed

The environment and tools proposed here are meant to fall on both ends of Figure 1 – and below the curve – i.e., be efficient. The first is a subset of the tools that, while being powerful enough for many high-level modeling needs, is also far easier to use and requires far less training than other existing tools. This subset of tools also provides an ideal environment for introductory education and training in the area of performance modeling and analysis. The second is an overall environment that can support the integration of a variety of modules, each of which may have been custom coded, in a cost

effective, efficient manner – with a strong view toward reuse. This environment is also ideal for more advanced education and training in performance modeling and simulation – with an emphasis on model reuse.

2.2 Summary of Most Important Results

The most important results are, of course, the definition, design and prototype development of the modeling and simulation tools and environments meant to fill certain voids and to address problems and shortcomings noted with existing tools and environments. Principle among these being the lack of: i) a simple to use yet effective performance modeling tool set and ii) an open, integrated, object-oriented performance modeling environment using a common programming language and promoting reuse. The former led to the Queueing Network Performance Engineering Tool (QNPET) and the latter to the Advanced Modeling, Analysis, Simulation and Engineering (AMASE) environment. These are both discussed below in Sections 2.2.1 and 2.2.2 respectively. Because of the more generic nature of a new method for parallel simulation incorporated in AMASE, that method is briefly discussed in Section 2.2.3. In addition, certain other generic results that also emerged from this research are noted in Section 2.2.4. (These results were an outgrowth of applying the tools prototyped to the performance analysis of some specific systems.) Finally, comments and discussion of continuing efforts (including additional documentation being written) are given in Section 2.2.5.

2.2.1 QNPET

One of the key results of this effort was the definition, design and prototyping of the Queueing Network Performance Engineering Tool (QNPET). (QNPET was initially referred to as the Modeling Analysis Simulation and Engineering (MASE) [PT-1] environment.) QNPET is an integrated tool set designed to provide high-level modeling capabilities easily accessible to the novice performance analyst but yet highly effective. It is built on a generalized queueing network paradigm similar to Q+ but differing from it in several ways. Besides being an integrated toolset, QNPET is less abstract and much easier for the novice to master than Q+; however it is also far less powerful – i.e., it is below and to the left of Q+ on Figure 1.

A very brief overview of QNPET is given below. The reader may find Appendix A of interest. It contains the lecture notes and viewgraphs used to introduce students to QNPET. (For more details on QNPET's features and use as well as detailed design documentation, refer to [PT-2] and other QNPET documentation references noted in Section 2.3.)

2.2.1.1 Overview of QNPET

The key tools comprising QNPET are the Editor, Consistency Checker, Analyzer, Simulator, Browser, Analyst's Assistant and Helper. The Editor provides a totally graphical means for building models. There is a limited, but powerful set of constructs that can be incorporated into models via "drag and drop". The Graphical User Interface (GUI) is designed to minimize the possibility of errors during parameter entry – e.g., choosing allowable entries from lists as opposed to typing. (For example, Figure 2 – also Viewgraph 7 of Appendix A - shows what a graphical description of a high level QNPET

model of the Army's SINCGARS communications system – including environmental effects - looks like.) Models constructed can be saved as complete models or as submodels or submodel templates. One of the constructs that can be dragged and dropped is a generic submodel, which is parameterized by providing a submodel name. Once built, models must be run through the Consistency Checker, which will identify any errors that prohibit the model from being loaded into the Simulator or Analyzer. For example, failing to provide a disposition – service time, routing, etc. - for transactions that arrive at a node.

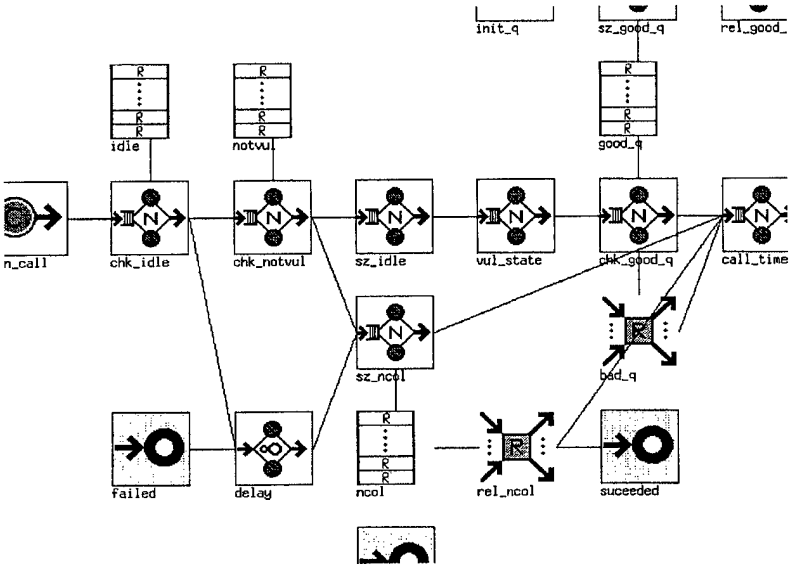


Figure 2 - High Level QNPET Model of a SINCGARS Radio Net

Once it has passed the Consistency Checker, the model can be loaded into either the Analyzer or Simulator. The Analyzer provides some additional checks that identify possible problems that might represent user errors. These include such conditions as unstable nodes, lost resources, unassembled packets, etc. Generally these are conditions that would not make sense for a simulation run seeking equilibrium solutions, but would be acceptable for a transient analysis. The Analyzer can also provide exact or approximate solutions for certain classes of models. (The class of models for which approximate solutions are available is currently being enlarged - e.g., to increase the applicability of certain infinite source approximations for finite sources [10], [11], [PT-3].)

Models loaded into the Simulator can be run in one of three modes: i) continuous mode – run till specified completion time; time step mode – run till the next specified time step, or event mode – execute only the next event. The time step given also serves to specify the times for statistics collection – independent of run mode.

In the Simulator, a monitor statistic (average total network sojourn time) allows the user to observe the evolution of the simulation, e.g., to determine when transients have died down. Comprehensive results are available directly from the Analyzer and Simulator in

ASCII format. However, a structured results file is also created for customized statistics viewing via the Browser. The Browser also allows access to other files associated with the model including the ASCII results files noted, the Consistency Checker and Analyzer generated results and the notes file which is associated with this model. (The editable notes file is accessible from the various tools and allows the user to keep comments about the specific model.)

The Analyst's Assistant is a (at present small) collection of evaluation and engineering functions for single node systems. For example, for a finite server loss system one can specify two of the following: offered load, number of servers, maximum blocking, minimum load on last trunk. The tool will then "engineer" the system to meet the two specified quantities and compute the rest along with other results. There is a GUI building tool for adding more functions, but, as we shall discuss shortly, a newer version of an Analyst's Assistant, prototyped in Java, may be a better alternative.

At present, the Helper is essentially a somewhat more concise version of the User's Guide - which also contains a limited amount of modeling guidance. (This aspect of the User's Guide is being significantly enhanced along with the development of more extensive tutorials.)

2.2.1.2 Education and Training with QNPET

As noted, QNPET has been incorporated into our introductory graduate course on performance modeling with great success. It takes very little class time to cover the basic use of QNPET for modeling, and after the students gain some familiarity, the most advanced features can be covered in a two hour combined lecture – lab. (Appendix A contains the lecture notes and viewgraphs used in a one-hour introductory lecture - followed by a one-hour lab.) The use of QNPET has also been incorporated into some of our more advanced performance modeling courses. Its use allows one to decouple the modeling aspects from the details of simulation coding. While there are several excellent products on the market that also do this, none can be mastered as rapidly and with such little class time. These factors also give it a high potential for use in the undergraduate curriculum - a prospect that we are currently investigating.

QNPET has also been incorporated into two of our short courses on performance modeling. These courses are offered to employees of local industry and government organizations. There has generally been excellent reaction to QNPET.

We are in the process of developing a more extensive tutorial for QNPET that will also provide modeling guidance to the user. This, along with course notes which incorporated QNPET as well as an overview of QNPET and its User's Guide and Reference manual will be posted on the web so that others can become more familiar with its potential use in education – as well as elsewhere.

2.2.3 AMASE

A second key result of this effort was the definition, design and prototyping of the AMASE (Advanced Modeling, Analysis, Simulation and Engineering) environment.

AMASE is an open, object-oriented environment consisting of a variety of integrated toolsets for modeling, analysis, simulation and engineering of computer / communications and other systems. (Early design of some of AMASE's architecture is given in [PT-4].) A primary objective of this environment is to promote software reuse, particularly the reuse of simulation models that can often be extremely costly to develop, in terms of both money and time.

AMASE, being a significantly more ambitious undertaking than QNPET is in more of a "prototype" stage than QNPET. The vast majority of the features discussed below are fully functional but some may require some user "hand editing" or command line entry, i.e., their use is not totally controllable via the GUI.

2.2.3.1 Overview of AMASE

AMASE is an integrated, open, object-oriented environment for modeling, analysis, simulation and engineering of computer / communications and other systems, designed with reuse in mind. Some of the key distinctive features of AMASE include: i) an integrated toolset supporting various aspects of performance analysis and engineering, ii) an open system view supporting the addition and integration of externally developed modules, iii) heterogeneous module support allowing integration of disparate modules, iv) modular model development support, v) hierarchical modeling support, vi) support for distributed simulation and vii) programmable nodes. (Hierarchical modeling support is an important feature for reuse, e.g., see [12].) While not all of these features are independent, their combination provides an excellent basis for fostering module reuse – one of the primary objectives of the AMASE environment.

Figure 3 shows the graphical representation of a simple model containing an Asynchronous Transfer Mode (ATM) switch. The ATM switch was actually one

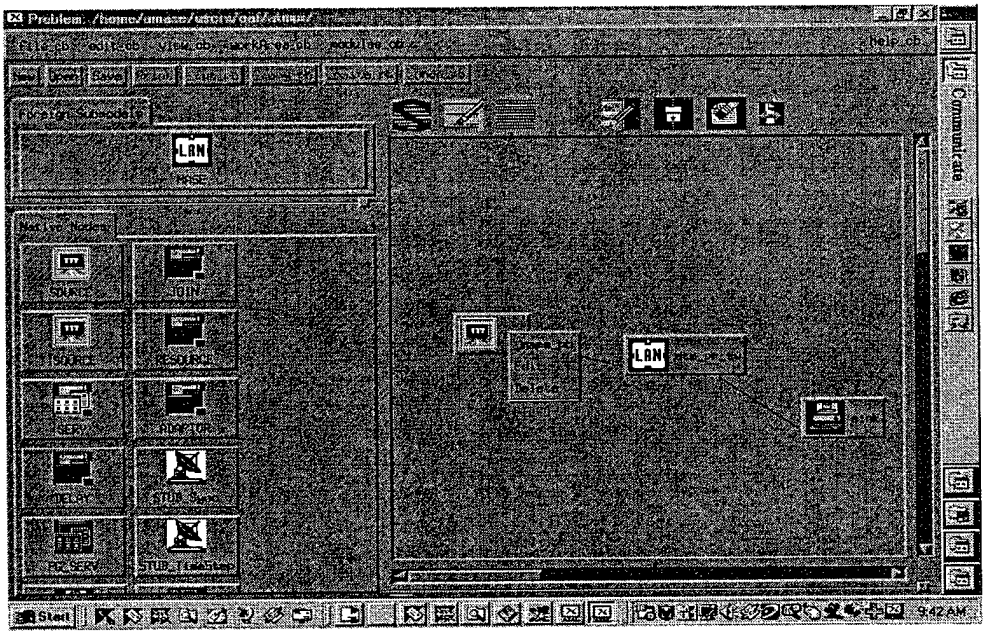


Figure 3 – AMASE Model with ATM Switch

constructed earlier with QNPET. Such models can readily be incorporated into AMASE as a submodel. While there are many similarities to the graphical model construction in QNPET, there are also many differences designed for more efficient model development. For example, Figure 4 shows the editing window that can be opened up from any node. The tabs allow one to quickly move among the various editable items, e.g., from

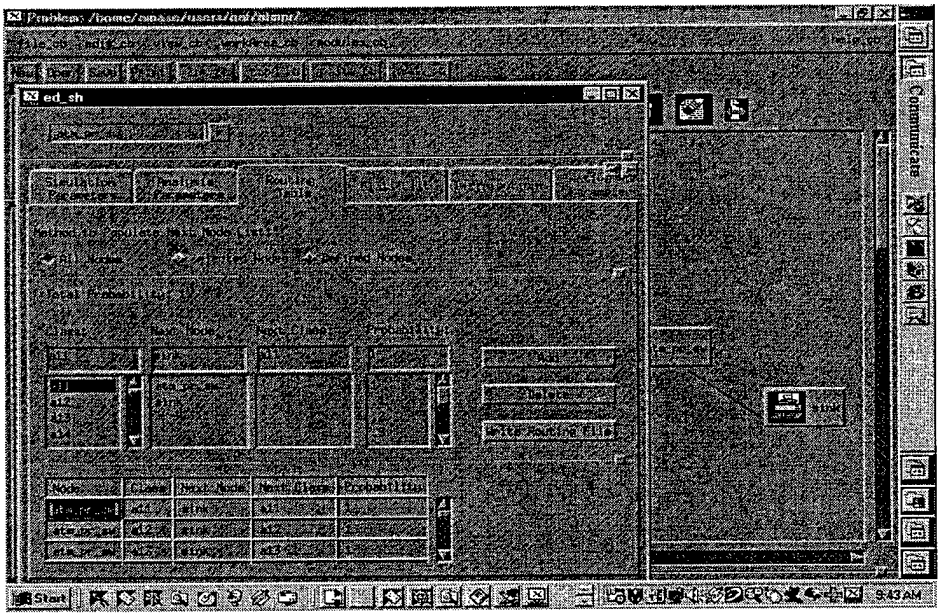


Figure 4 – AMASE's Universal Editing Window

specifying routing to distribution entering, etc. Also, the routing shown is global, i.e., one can scroll to find the routing designated at any node without having to switch to an editing window for that node. The drop down selection menu in the upper left hand corner allows the user to switch the node being edited at any time.

The AMASE environment provides the mechanism for a user to build a "supermodel" (federation) by making use of the combined capabilities of an integrated set of simulation modules (federates) from its library – as well as providing a variety of tools and utilities. AMASE specifications include minimal interface requirements for modules to be added to the system library. Such modules are registered with AMASE at a given compliance level and as belonging to a given communicating class. In this way, AMASE can integrate a variety of non-homogeneous modules so that the user can make effective use of their combined capabilities at a level commensurate with the problem at hand. (The AMASE environment is consistent with HLA requirements – see Appendix A.)

The simulation control environment (see Figure 5) supports multiple simulation runs simultaneously. While all may be actively running, one is designated as the "active" simulation, implying that it can be actively controlled, e.g., paused, stopped, mode changed, etc. The user may select any simulation to be active at any time. To start a new simulation, the user opens the "new simulation" window (see Figure 6) where the new simulation can be instrumented. Besides choosing between various modes, the user can

launch a specific number of simulations with identical parameters – but different random number seeds- at the same time, e.g., for ensemble averaging. The Browser provides support for viewing and some analysis of the collection of runs.

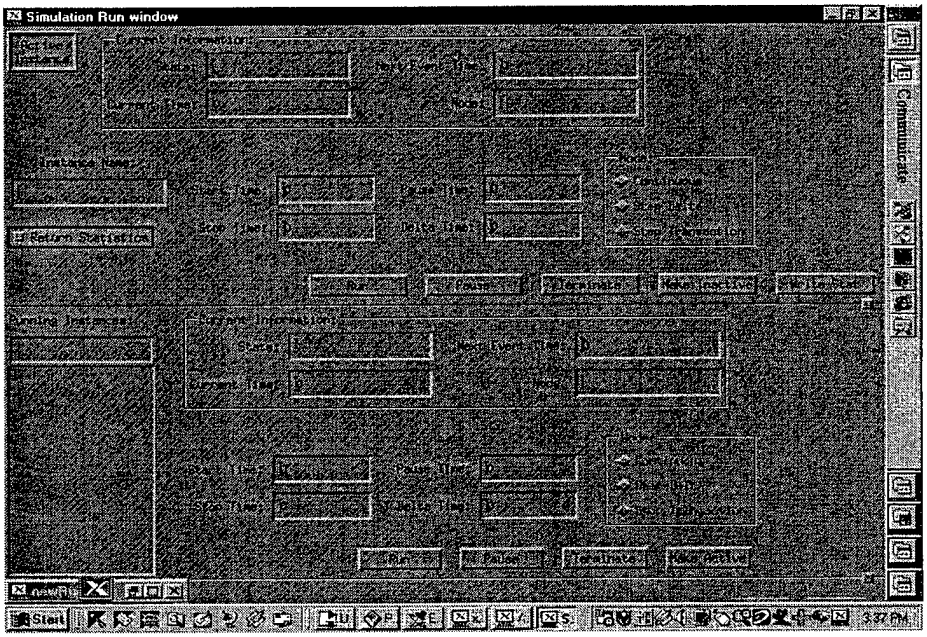


Figure 5 – AMASE's Main Simulation Control Window

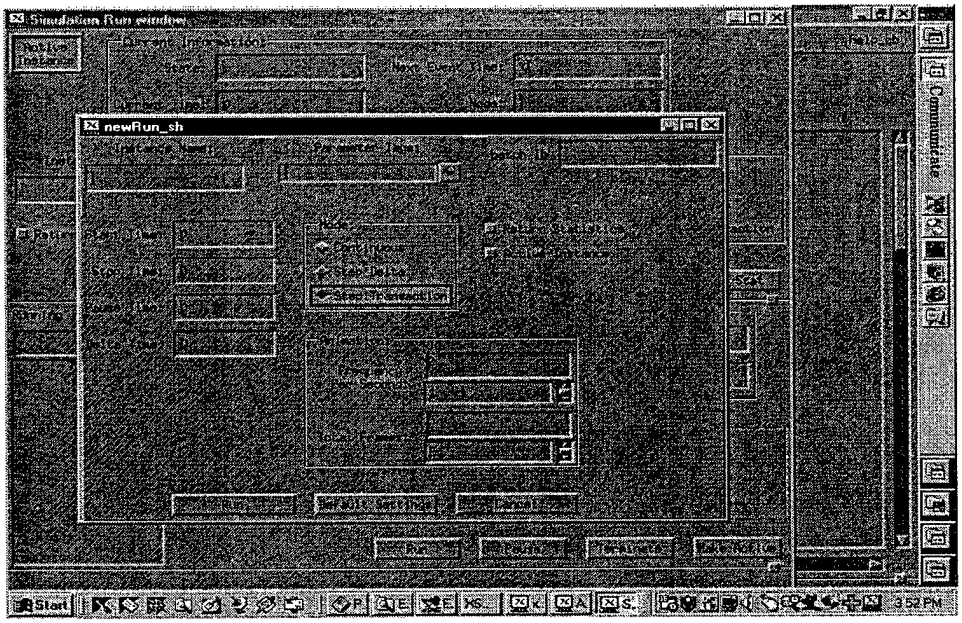


Figure 6 – Window for Launching New Simulations

2.2.3.2 Promoting Reuse

Below we list, along with a brief discussion, some of the key AMASE features designed

to promote software reuse. (In [PT-5] specific qualitative criteria for evaluating reusability, based on [13], [14] and taking into consideration [15], [8] are given and, using these, AMASE is evaluated for reusability.)

Flexible Module Interface Specification:
In order to allow various externally built modules to be "plug compatible", it is necessary to specify certain interface requirements that must be met. The challenge is to impose the least restrictions on such modules while ensuring easy interoperability. AMASE achieves this in a variety of ways. Interface specifications basically have two facets. One is concerned with the services that an externally developed module will provide and the other with the messages that the module exchanges. These are, of course related; however they do have their separate issues.

In terms of services, AMASE supports the use of "conformance levels". At the highest level, for a simulation module, this essentially corresponds to controllability. A simulation module must, of course, be created and initialized; but in addition, the simulation controller must be able to perform primary control functions such as asking the module to simulate up to a given time, until its next external event, etc. The interface specification that accomplishes conformance at this level consists of five well-defined functions that provide the minimal service set at this compliance level. By supporting more services, a module can raise its compliance level and thereby increase its capabilities and hence its desirability for reuse. Additional functionality, for example, can include sending statistics and animation messages for the system to display. Reusability is also enhanced by providing certain services that externally developed modules can take advantage of - see below.

All modules must support the basic transaction class, consisting of seven fields, which is the main mechanism used for communications between modules. Modules that support only this transaction class form the basic "communicating class" – all modules are capable of exchanging information at this level. In addition, when a module is registered with he system, it must register its communicating class. This can be one of the existing communicating classes, or the module(s) can create a new one. A communicating class forms a collection of modules that exchange the same transaction types, i.e., in addition to the basic transaction fields, the derived transactions for the class have the same supplemental part. Reusability is also enhanced by providing a mechanism for modules in different communicating classes to communicate – see below.

With these features, externally developed modules can be used in AMASE to build models consisting of an interconnected collection of such modules. With virtually no effort, they can make some of their services available for use by other model developers who wish to connect them with their own modules. By making use of AMASE utilities (discussed below) they can further enhance the functionality that they provide to others. Finally, with a bit more effort, they can make all of their services available to others.

Communications Facilities for Interconnecting Heterogeneous Modules
There are essentially two facets to this feature also. One is concerned with the

interconnection of modules of different communications classes that therefore have derived transaction components that are not recognizable by each other – nor are they usable. The other facet is the ability to interconnect modules on different platforms written in different languages.

Since all modules support the basic transaction class, all modules can make use of a set of basic services that any other module provides. However, information contained in the supplemental part of the derived transaction that might be needed to make use of the full services of modules in a given communicating class is preserved by the system. AMASE recognizes the disparity between the communicating classes of a sending module and a receiving module. The supplemental portion of a transaction that will not be recognized (and hence cannot be used) by the receiver is removed by AMASE and stored. When that specific transaction is again sent to a module that would recognize the supplemental part unique to this communicating class, the derived transaction is reattached.

In addition, the user can make use of programmable adapters to allow a richer set of services to be accessed. The adapters basically convert the incoming transaction to the transaction class of the receiving module. Obviously, how this is done depends on the particular model being developed - hence the support for programmability.

From AMASE's view all modules, whether a simple element or a complex full model are viewed as submodels. There are two types of submodels, standard submodels and stubs. From AMASE's perspective they are treated identical except that during initialization, stubs are given the location where they are to create the intended submodel. The stub establishes the communications facilities and initializes the remote submodel – which could be a full model – see below. To the remote model's AMASE controller the stub looks like the user interface while to the local AMASE controller it appears to be just another submodel. Note that the remote machine could actually be the same platform that the local controller is running on. In the simulation context, the stubs can currently run in one of two modes: i) synchronous, where no potential performance gain for using multiple processors can occur. (This is useful when it is merely platform incompatibility that matters) and a novel mode we term ii) Maximum Allowable Error [PT-6], where the stub allows the remote model to run asynchronously, but hides this fact from the local controller. In this case a maximum error for time adjustments is given and enforced by the stub. (This is usefully when it is known that errors cannot really occur or can be prevented, or when small errors in event time rearrangement are acceptable in order to speed up computation time. This method of distributed simulation is discussed further in the next section.) In addition, support for a rollback mode can readily be added, however, its usefulness would depend on the existence of modules that support rollback.

With these features, modules built on different platforms can be readily interconnected. Also, in most cases, models that where not built with AMASE in mind can be adapted for remote stub control. This amounts to putting a "wrapper" around the model that supports the five basic submodel control functions. (Having been built before AMASE, QNPET's design did not anticipate the interface requirements that AMASE would impose. However, QNPET models can be readily "wrapped" and included in AMASE as

submodels – as was done with the ATM switch model discussed above.)

Automatic Submodel Creation

Any model built with AMASE (i.e., an interconnection of heterogeneous modules) can be made to run as a submodel. This is accomplished by having a stub create the model – the stub will then function as a submodel to the local controller. (Thus there are times when it makes sense to have a stub create its model on the same machine as the primary controller or even in the same process space.) This feature is important for promoting reuse. Once a collection of modules has been put together to accomplish a specific task (set of services) it can be saved as such and reused whenever those same sets of services are needed.

Common Model View with Plug Compatibility

Once a model has been created (graphically) by using various modules and parameterizing them, one may want to apply a variety of tools to e.g. evaluate a scenario. In addition to being able to simulate the model at hand, AMASE provides analytic approximations to evaluate performance characteristics of the system. Providers of modules can provide analytic approximations if desired, but AMASE allows the user to characterize the performance of modules by one of the existing stochastic service centers for which analytic approximations already exist. This common model view, along with the stub concept can be used to support the interconnection of actual system modules (as opposed to simulations of them). Thus a model of a database system could be replaced at the appropriate time with the actual database system – receiving its queries via the stub. Indeed, one could even include a person in the loop. To view this another way, a detailed simulation model of a key communications facility could be used in a variety of contexts, e.g., to study the facility in isolation with inputs and outputs provided by analytic approximations, to study a system where the facility simulation is one of several modules or by interfacing to real system components.

Utilities

Several utilities are available which builders of modules can make use of both to help them to accomplish their own task and also to enhance the services that they provide to others. We list just a few here:

i) **Animation of Transactions** – module builders can send out transactions with their animation bit on. AMASE highlights the module sending and receiving the transaction and also display the entire transaction contents for the user. This can be extremely useful for debugging and for reinforcing a potential user of a module as to its actual operations.

ii) **Statistics Transactions** – modules can send statistics to the system for display to the user at regular intervals.

iii) **Accessible Statistics** – by making the names of a files where appropriate comma delimited data are stored, known to the system, such statistics files can be handled as if they were created by a library module, i.e., viewable in the Browser and subject to statistical analysis.

iv) **Adapters** – AMASE supports the concept of adapter modules in which the user can provide customized versions of "converters" to translate transactions from one communicating class to another.

v) **Module Library** – the model library contains utilities, e.g. to manage shared resources, complex split / join capabilities, etc.

2.2.3.3 The Maximum Allowable Error Mode for Distributed Simulation in AMASE
As noted, there are basically three ways that one can consider running AMASE in a distributed environment. A major result obtained here was the introduction of a novel method of running distributed simulations that has broader applicability. Methods for achieving parallelism in discrete event simulation are generally considered to fall into two categories: i) conservative methods where no events are allowed to take place out of (time) order and ii) optimistic methods where events may occur out of order, but a mechanism exists for correcting the order. The conservative methods generally rely on being able to predict next event times in advance, or at least bound these. Optimistic methods often employ a rollback mechanism whereby the controller (which is responsible for simulation synchronization), after detecting an event out of order, can cause all modules that are potentially affected to "roll back" their clocks (and states) to a previous correctly synchronized time. There are obviously many tradeoffs involved and various implementations have been studied in great detail. One of the main issues raised in [16] is highly relevant here. Namely, there are a great deal of tools and expertise available for building serial simulators, and indeed, one often finds that many of the simulation pieces that might be needed to address a problem are currently available in serial simulations built earlier. Indeed, as noted in [16] (with a reference to [17]) there are many cases in government and industry where just being able to interface various existing simulations would be a welcome advance - any parallelism would be an unanticipated bonus. Indeed, this was the primary motivation for the development of distributed simulation capabilities presented here.

The method introduced in [16], referred to as U.P.S. (Utilitarian Parallel Simulator) is meant to link together serially designed simulators with minimal addition effort. Of course, since U.P.S. is a conservative method, a mechanism must be implemented for extracting future event time bound information from the modules. U.P.S. takes advantage of being able to "mix" the synchronization protocols used by various modules to take advantage of known structures. The net result is rather good speedups for a broad class of examples - where event bounding is or can be made available.

We also want to minimize the additional effort to be imposed on serial simulations; however, we take an approach which is quite different in nature, and, admittedly also in its applicability. In particular, we allow events to be processed out of order (in a controlled manner) without having a mechanism for correction, e.g. via rollback. This clearly eliminates the need to predict future event time bounds, but obviously it can introduce errors. Our method is designed to control such errors. The basic objective of this new method is to coordinate the processing of various simulation modules, distributed over a variety of processors, and keep the primary event time errors below a

specified value - the Maximum Allowable Error (MAE). The term "primary error" refers to the errors that are visible to the controller, i.e., an event, E_1, that was supposed to happen at time t_1, did not make itself known to the appropriate processor's controller till time $t_2 > t_1$. The system will ensure that the event, E_1, will actually occur no later than t_1 + MAE. Maintaining this maximum allowable error does not preclude rather large errors from being induced into the simulation. For example, this is probably not a good protocol to employ for studying real time control of nuclear reactors. However, there are many cases where having small errors in arrival times of transactions has very little impact. What is small? That of course depends on the application, and indeed, with our method, as with many others, the most gain will occur when we can take advantage of the physics of the problem at hand. For example, in a virtual reality simulation for a given application, delays of less than 10 ms. for stimuli that a person in the loop sees may be irrelevant. Note that it is perfectly acceptable to set the maximum allowable delay to 0, in which case we would expect to run the simulation in lock step mode, i.e., totally serially with all events processed at the correct time. The result would be no speed up at all (and no errors); indeed there would more likely be a "slowdown" (speedup < 1) due to the communications overhead needed to coordinate modules distributed over several processors. As noted, this may be acceptable in certain circumstances where the integration of two or more simulations is the primary concern, not parallelism. (The MAE method is described in more detail in [PT-6] where some quantitative results are included.)

2.2.4 Other Important Results

The tools noted above have been used in a variety of performance analysis studied that have resulted in some important results in their own right. Perhaps key among these was the discovery that holding times of connections in a communications network can have an impact on the resulting performance characteristics of bursty traffic. It has long been recognized that, for example, blocking in circuit switched networks depends on the burstiness of the traffic [18]. Generally, the burstier the traffic, the larger the resulting loss. Since blocking for Poisson traffic is known not to depend explicitly on holding times, but only on the resulting offered load (arrival rate times holding time) the same has generally been assumed for bursty (peaked) traffic in all standard engineering practices. However, as shown in [PT-7], two traffic parcels with the same burstiness (peakedness) and having the same offered load, but with different holding times, will see different blocking. In general the parcel with the larger holding time sees less blocking! An important application of this result is to the provisioning of circuits which are shared by long holding time Internet traffic and standard voice calls [19].

These tools were also used in a variety of other studies. One was to investigate the impact of using Open Shortest Path First (OSPF) in an internet environment composed of interconnected multiple access networks (e.g., SINCGARS). The results showed that attempting to initialize such a network too rapidly could have a significant negative impact on performance and an impact that would repeat periodically [PT-8].

2.2.5 Comments and Continuing Efforts

Research and other efforts related to both QNPET and AMASE are continuing at Monmouth University's Simulation and Modeling Lab (SIMLAB) and elsewhere in the Center for Technology Development and Transfer. We fully expect such efforts to continue in the foreseeable future,

As noted, we are currently enhancing QNPET documentation by adding a comprehensive tutorial on its use for computer / communications (and other) systems modeling. The aim is to eventually have a complete educational package that could be used asynchronously as well as in traditional course settings. Besides traditional publications, we expect to shortly have sufficient information on the web to promote awareness of QNPET's in relevant communities. Because of the ease in adding functionality to AMASE's version of the Analyst's Assistant, it will be taken as the standard for continued upgrading and will eventually replace QNPET's current Analyst's Assistant.

Completing documentation for AMASE is an important aspect of our continuing efforts as is publicizing both AMASE and the key concepts that it demonstrates.

2.3 Publications and Technical Reports

[PT-1] A.A. Fredericks and W. Jing, "MASE -- an Integrated Environment for Modeling, Analysis and Engineering of Computer / Communications Systems", GLOBECOM '95, pp. 1688 - 1692.

[PT-2] QNPET (MASE) User's Guide and Reference Manual

[PT-3] A. A. Fredericks, "A new Look at the Infinite Source Approximation for Finite Sources", ", Monmouth University, Center for Technology Development and Transfer, Technical Report.

[PT-4] W. Jing and A. A. Fredericks, "An Open Object-Oriented Simulation System for Communications Networks", GLOBECOM '95, pp. 1847 - 1951.

[PT-5] A. A. Fredericks, "Promoting Simulation Model Reuse with the AMASE Environment", Monmouth University, Center for Technology Development and Transfer, Technical Note.

[PT-6] A. A. Fredericks and Yan Li, "The Maximum Allowable Error Mode for Distributed Simulation in the AMASE Environment", Monmouth University, Center for Technology Development and Transfer, Technical Report.

[PT-7] A. A. Fredericks, "Impact of Holding Time Distributions on Parcel Blocking in Multi-Class Networks, with Application to Internet Traffic on PSTN's", Teletraffic Engineering in a Competitive World, Elsevier Science, B. V., Amsterdam, The Netherlands, Volume 3b, pp. 877-886.

[PT-8] A.A. Fredericks and W. Bastian "Performance Impact of the Initialization Interval for Certain Networks Employing OSPF Routing", Teletraffic Contributions for the Information Age, Elsevier Science, B. V., Amsterdam, The Netherlands, Volume 2a, pp. 711-720.

Other QNPET Documentation

W. New, "System Overview and Design Documentation for QNPET's Simulation Tool", Monmouth University, Center for Technology Development and Transfer, Programmer's Note.

K. De Bose, "System Overview and Design Documentation for QNPET's Analysis Tool", Monmouth University, Center for Technology Development and Transfer, Programmer's Note.

Other Documentation: As noted, additional documentation for QNPET, including tutorials and course notes, etc. – including web publication – are in preparation. Full documentation (including submissions for publication) of the AMASE environment together with a user's guide will also be forthcoming.

2.4 Participating Personnel and Advanced Degrees earned while on project.

Table 1 below shows participating personnel by category – full or part time researcher or student research assistant. Note that during the time of the Grant, some persons moved from one category to another and therefore appear in more than one place in the table. Also note that Monmouth University does not have any PhD programs; hence a Masters degree is the highest degree we offer.

Full Time Researchers	Part Time Researchers	Research Assistants
A.A. Fredericks	W. Bastian	P. Battista**
V.V. Basapur	A. A. Fredericks	J. Ding*
P. Batista	Y. Li	L. Kataria*
K. De Bose	W. New	V. Lakshmi*
Y. Li		V. Sanjivi*
M.K. Moskal		M. Sherman*
W. New		S. Sheth
		V. Vinayak*
		J. Xiang*
		L. Yang*
		J. P. Yu*
		P. Yu*

Table 1 – Participating Personnel

* Received M. S. Degree

** Received B. S. Degree

3. Report of Inventions

None at this time.

4. References

[1] OPNET MODELER, OPNET Technologies inc. (MIL 3, inc.), Washington, DC.

[2] NETWORK, CACI Products Company, La Jolla CA.

[3] Bones, Comdisco Systems Inc., Foster City, CA.

[4] COMNET, CACI Products Company, La Jolla, CA.

[5] SIMSCRIPT, CACI Products Company, La Jolla CA.

[6] MODSIM, CACI Products Company, La Jolla CA.

[7] Q+ (Performance Analysis Workstation), Bell Laboratories, Lucent Technologies, Holmdel, NJ.

[8] A. M. Law and Michael G. McComas, "Simulation Software for Communications Networks: The State of the Art", IEEE Communications Magazine, March 1994, pp. 44 - 50.

[9] Emilie T. Saulnier and Betty J. Bortscheller, "Simulation Model Reusability", IEEE Communications Magazine, March, 1994, pp. 64 - 69.

[10] W. Whitt, "Open and Closed Models for Networks of Queues", AT&T BLTJ, Vol. 63, No. 9, Nov. 1984, pp. 14 – 31.

[11] A. Fredericks, "An Approximation Method for Analyzing a Virtual Circuit Based LAN – Solving the Simultaneous Resource Possession Problem", Teletraffic Analysis and Computer Performance Evaluation, O. J. Boxma et al, ed., Elsevier Science Publishers B. V., (North Holland), 1986, pp. 63 – 74.

[12] F. E. Cellier, Q. Wang and B. P. Ziegler, "A Five Level Hierarchy for the Management of Simulation Models", Winter Simulation Conference, 1990, pp. 55 - 60.

[13] R. Prieto-Diaz, "Implementing Facetede Classification for Software Reuse", Communications of ACM, vol. 34, No. 5, pp. 88 - 99, May, 1991.

[14] Oryal Tanir and Suleyman Sevinc, "Defining Requirements for a Standard Simulation Environment", IEEE COMPUTER, February 1994, pp. 28 - 34.

[15] Kenneth D. Shere and Rachelle A. Carlson, "A Methodology for Design, Test, and Evaluation of Real-Timje Systems", IEEE COMPUTER, February 1994, pp. 35 - 48.

[16] Nicol, D. and P. Heidelberger, "Parallel Execution for Serial Simulators", ACM Transactions on Modeling and Simulation, July 1996, Vol. 6, No. 3, pp. 210-242.

[17] Wilson, A. L. and R. M. Weatherly, "The Aggregate Level Simulation Protocol: An Evolving System", 1994, Proceeding of the 1994 Winter Simulation Conference, IEEE Computer Society Press, Los Alamitos, CA, pp781-787.

[18] R. I. Wilkinson, "Theories for Toll Traffic Engineering in the USA", BSTJ 35, pp 421, (1956).

[19] E. E. Cohen, A. A. Fredericks and C. D. Pack, "The Internet and the Switched Telephone Network – A Troubled Marriage", Teletraffic Engineering in a Competitive World, Elsevier Science, B. V., Amsterdam, The Netherlands, Volume 3a, pp. 23 – 32.

Appendix A

Introduction to the Queueing Network Performance Engineering Tool (QNPET – MASE) - Lecture Notes and View Graphs

This appendix contains the lecture notes and view graphs used to introduce students to QNPET. It is followed by a lab session were the students follow the example given in the view graphs to create their first model.

Lecture 5

5. The Queueing Network Performance Engineering Tool (QNPET - MASE)

Both simulation and analytic modeling are often employed during the various phases of a system's lifetime - e.g., definition, design, development, engineering and operations. A primary objective of such modeling is usually to insure that performance objectives are met in a cost-effective manner. Often a variety of simulation and analytic tools are used, but invariably there is no relationship between any of the tools. To address this problem (as well as others, including model reuse), an open, object-oriented environment consisting of a variety of integrated toolkits for modeling, analysis, simulation and engineering of computer /communications systems is being developed at Monmouth University. QNPET, an integrated toolset built around a generalized queueing network paradigm is the first toolset in that environment.

The earlier name for QNPET was MASE (Modeling, Analysis, Simulation and Engineering); we will use the terms interchangeably.

5.1 Background

Performance modeling and analysis provides an important means for ensuring the cost-effective design, engineering and operations of computer/communications systems. Appropriate performance modeling and analysis techniques can provide quantitative insight into system performance that would otherwise be difficult, too expensive or even impossible to obtain by other means. This is especially true for very large, complex systems and in particular when performance under potentially adverse conditions is extremely important. Most organizations have used, and continue to use a diverse set of simulation and analysis tools to meet their modeling needs. Often multiple tools are needed to employ hierarchical modeling for addressing different performance issues. For example, at a very early stage, one might use simple analytic formulae from queueing theory to roughly size components of the system. For network issues, one might use Q+ [1] to do some high level modeling and turn to a more comprehensive simulation tool such as OPNET MODELER [2] to study protocols in detail. When specialization and additional detail are needed, a simulation language such as SIMSCRIPT II.5 [3] might be used. (For a discussion of the state-of-the-art of communications network simulation tools. including many available packages, see [4]; for a discussion of the importance of hierarchical modeling see [5].) Generally, these tools do not -communicate" with each other so that, at the very least, additional data entry (subject to user input error) is needed to use multiple tools. More importantly, different tools usually have different basic modeling constructs so that comparison of results is difficult and limited at best, impossible at worst. The net result is often higher costs and longer lead times for model development, costly and time consuming (and often deficient) verification/validation procedures and great difficulties in interfacing various subsystem models. Under the auspices of the Army Research Office (ARO), an effort has been undertaken at Monmouth University's Simulation and Modeling Lab (SIMLAB) to address some of these problems by defining and developing an integrated, open, object-oriented

environment for modeling, analysis, simulation and engineering of computer/communications systems. QNPET (MASE) is the first toolset in that environment. (See [6], an expanded version of these notes that includes example applications.)
QNPET (MASE) is an integrated toolset for modeling, analysis, simulation and engineering of computer/communications (and other) systems. QNPET (MASE) uses a generalized queueing network paradigm to provide a high level conceptual view of the systems to be modeled. From this perspective, its simulation tool is quite similar in conceptual content to Q+ [1,7]. However, since QNPET (MASE) will be the "simplest" of the toolsets in this evolving environment, it has been designed with the view of simplicity of use over complexity of functionality. More importantly, QNPET (MASE) is an integrated toolset that provides analytic solutions (approximations where appropriate), engineering support tools and a variety of utilities e.g., model consistency checking. These features make QNPET (MASE) an ideal candidate for use in a variety of courses, both at the graduate and undergraduate levels.

5.2 Overview of the QNPET (MASE) Toolkit

The toolkit for QNPET (MASE) consists of seven "tools": Editor; Consistency Checker; Analyzer; Simulator; Engineer's Assistant; Browser and Helper. The following gives a brief overview of each of these tools:

Editor: The Editor provides a graphical user interface to build and parameterize a model for analysis and/or simulation. Icons representing constructs useful for modeling computer/communications systems are provided - these are discussed in some detail shortly. Once built, a model can be saved as a "master" model or as a submodel. A master model is a runnable, complete model. Submodels cannot be run independently, but rather must be embedded in a master model via the Submodel node construct provided by the Editor - see below. When using the Editor, the user can create a set of "notes" for the model being worked on and record any information that might be helpful, e.g., the purpose of the model, assumptions made, etc.

In an effort to assist the user in parameterization as well as to minimize user input errors, whenever possible the user is shown allowable inputs to choose from. For example, for routing, the user selects the desired transaction(s), chooses routing, and is presented with a form for entering routing, including a list of acceptable next nodes to select from.

Consistency Checker: This tool checks models for consistency to ensure that they are loadable by the Simulator and Analyzer. For example, it checks to see that the model is completely specified, i.e., the system knows what to do with every transaction that can arrive at a service center. An appropriately consistent model is "marked" and only models so marked can be loaded into other tools. In addition, a variety of checks are made which will not result in the model being marked inconsistent; but will result in a warning to the user. For example, the utilization at all the nodes due to open chains is checked (which is sufficient to determine "load' stability). Also, if passive resources are used, checks are made to see that seized resources will be properly released by

transactions, prior to their exiting the system. Various split / join node pairs are also checked to see if all transactions emanating from a spilt node do indeed arrive at the designated join node. The user is only warned of any adverse conditions since it is meaningful to study the transient effects of such systems with the Simulator.

Analyzer: The Analyzer can estimate performance measures for the queuing network using several approximation techniques. These include enhancements to the closed chain approximations given in [8, 9] which result in accurate predication of utilizations (which could be 1 for closed chains in a stable system) and throughputs in addition to response times. The Analyzer also forms the basis for many of the checks performed by the Consistency Checker. Note that while the Analyzer will load an unstable model, it, of course, cannot solve it, but rather must inform the user of the situation. The user can edit many of the parameters from the Analyzer (e.g., make the model stable) and evaluate the results. These changes are local and don't propagate to the main model.

Simulator: This tool can simulate the resulting queuing network model outputting both transient and final (e.g., equilibrium) results. A variety of simulation controls are provided (e.g., a "delta" time for collecting output statistics). The simulation supports a variety of constructs useful for modeling computer/communications systems including data multiplexing, flow control, broadcasting, process synchronization, resource management, etc. - see below. A monitor statistic is displayed allowing the user to observe the evolution of the simulation, e.g., to determine when transients have died down. The simulation can be paused, restarted or stopped at any time. Like the Analyzer, the user can make "local" edits to the model loaded in the Simulator.

Engineer's Assistant: The Engineer's Assistant provides support for a variety of single service center queuing paradigms. For example, M/M/1 queue, Erlang B etc. which are "exact" solutions plus approximations for others, e.g., Hayward's approximations for peaked traffic. Each of these is accessed via a common view, form entry system. In addition to evaluation capabilities, many engineering problems are supported. For example, for a given offered traffic, one can specify cell loss, and have the number of buffers needed determined. A graphical interface is provided to allow users to add their own functions to the system.

Browser: This tool allows the user to browse previously calculated results files from any of the tools, "notes" files created for models, etc.

Helper: This tool provides access to a limited version if the user's guide and reference manual. The latter, available via a hypertext file, provides extensive help on the use of QNPET – it will also be upgraded to include more extensive "modeling" help.

5.3 QNPET (MASE) Constructs

As noted, QNPET (MASE) uses a standard generalized queueing network paradigm. Under this, the fundamental element corresponding to work and/or control is the transaction. Users only see a transaction through its user specified class name.

Source Nodes: There are two types of source nodes:

i) **I-Source Node**: An I-Source node represents an initialization source which is used to initialize a given node with desired transactions, e.g., for initializing a closed chain or "preloading" the system.

ii) **C-Source Node**: A C-Source node represents a "continuous source", i.e., one where an arrival process for transactions with given classes is specified. A simple list choice and form entry allows for the specification of routing and parameterization of source traffic streams with the desired interarrival distributions, distribution parameters and classes.

Server Nodes: The are two basic server nodes:

i) **Delay Node**: A Delay node represents an infinite server or delay node at which the distribution for the delay is specified as well as routing from this node to other nodes. This construct is distinct from an N-Server node to emphasis the conceptual difference between modeling delays and actual service centers - often a point of confusion with students.

ii) **N-Server Node**: An N Server node represents a set of N homogeneous servers. Non preemptive priorities are supported at this time. In addition, the user can specify a number of resources that a transaction needs (from a specified resource pool) in order to proceed. Such resources can be used to represent windows in a window flow control, memory in a computer system, a database lock, etc. Alternate routing may be specified for transactions that do not find the needed resources available. (This can also be done for transactions that find all buffers full.) Again, all parameterization is done via choosing from lists and form entry to reduce the possibility of user input errors.

Routing Nodes: In addition to specifying routing at other nodes there are two nodes where only routing is specified.

i) **(Standard) Routing Node**: A standard Routing node represents a node where only routing is specified - no service. E.g., to distribute traffic from a source or other node, the traffic can first be directed to a Routing node. The routing is probabilistic by class with class changes allowed. (While this standard routing could be specified at other nodes, it is sometimes advantageous to isolate the routing function.)

ii) **Distributor Node**: A Distributor node provides routing in a different way, totally class independent. The user specifies a set ($Node_1$, Num_1; ... $Node_k$, Num_k) of nodes to route to ($Node_i$,) and the number of transactions (Num_i) to route there. Routing is done "cyclically" where the first Num_1, arrivals are routed to $Node_1$, etc. Typical examples include broadcasting/multicasting support (when combined with a Replicate node - see below) and source traffic distribution.

Split / Join Nodes

Split nodes allow a single entering transaction class at the input to generate multiple transactions of various classes at the output. There are three varieties of Split nodes that are designed to meet common needs in modeling computer / communications systems, e.g., to model multi-layer protocols in a natural way. Some of these may be combined with an appropriate Join node to accomplish a desired task. The corresponding Join nodes are optional. A Join node could also be used first with or without a Split node. For simplicity of construction, a user need only choose the option of having a corresponding Join or Split node; the system creates and parameterizes it automatically. By using Split / Join nodes appropriately, one can obtain arbitrary point-point delays. The following discusses the various "flavors" of Split / Join nodes, five in all.

i) **Fragment / Assemble Nodes**: At the arrival of a single specified class, say m, a Fragment node generates a set of departure transactions of the form (c_1, n_1; .... c_k, n_k) where a pair (c_i, n_i) denotes n_i transactions of class c_i, are to be generated. Optionally, one may specify that an Assemble Node should be created. At that node, when the transactions that originated at the corresponding Fragment node arrive, they are grouped together and "assembled" into a single
transaction, m - when all of the relevant parts are collected. It is the specific transactions that were part of the original fragmenting that are assembled - their class name on arrival is irrelevant A typical application is fragmenting and reassembly in data communications networks.

ii) **Fork / Synchronize Nodes:** A Fork node is similar to a Fragment node except that the arriving transaction is assumed to have "forked" the transaction set (c_i, n_i; .... c_k, n_k) and so it itself still persists - i.e., it is routed out of the node with the other transactions that are generated. The corresponding (optional) Synchronize node behaves like the Assemble node described above. A typical application is the synchronization of processes in a computer system.

There are some subtle but important implementation differences between these two Split / Join node pairs. E.g., in the case of a Fragment / Assemble pair, the parent transaction is "suspended" on arrival (with any resources it holds) and "revitalized" when the appropriate transactions are reassembled. For a Fork / Synchronize pair, the original transaction arriving at the Fork node maintains its identity, including resources that it may hold throughout its network excursion (it may release them at any time) until it arrives at the corresponding Synchronization node.

iii) **Multiplex / Demultiplex Nodes**: To some extent this pair is the complement to the Fragment / Assemble Node pair. At a Multiplex node, one specifies a combination of transactions, (c_1, n_1; .... c_k, n_k), to be "multiplexed" into a single transaction, say class m. Optionally, one can specify a Demultiplex node where the arriving transactions (e.g., the m's) that were previously multiplexed are demultiplexed resulting in a set of =actions, (c_1, n_1; .... c_k, n_k), leaving this node. A typical application is multiplexing demultiplexing communications lines.

Two Split nodes do not have corresponding Join nodes.

iv) **Replicate Node**: At a Replicate node, an arriving transaction is replicated into a specified number of transactions with specified class names, e.g., for different routings. A typical application would be multicasting/mailing lists.

v) **Resource Split Node**: At a Resource Split node, an arriving transaction, say m, that is carrying a set of resources, can split off another additional transaction, say m'. The user can then specify which resources originally with m should stay with m and which should be transferred to m'. Typical applications are in multilevel protocol window flow controls and other resource management schemes such as computer memory management.

Other Construct Nodes

Sink Node: A Sink Node accepts routing to it and destroys arriving transactions - after accumulating statistics.

Resource Node: At a Resource node one specifies the resource provisioning process for the resource requirements given (optionally) at N Server nodes. One can have a fixed pool of resources (that must be returned to be reused) or specify a rate to add resources. The former are useful for modeling window flow controls and other passive resources. The latter can be used to model rate control schemes such as a leaky bucket algorithm

Submodel: A Submodel node corresponds to a previously defined and saved submodel built within the MASE system. It is parameterized by specifying the file name for the submodel desired and then specifying routing for transactions that leave the Submodel node. (One, of course, generally also routes to the Submodel node) When a standard model is built, the user can ask to save it as a submodel, either a template or a usable submodel. This will require the specification of a node to act as an input port and a node to act as an output port. To include a usable submodel, the user places a system generated Submodel node icon into the model and, as part of the parameterization process, the user specifies the name of the submodel (a list of available submodels is supplied). That submodel (file) is now linked to this model and cannot be reused until it is released. If a submodel template is chosen for inclusion, the system first makes a copy (which is then usable) and links this to the current model. Templates are reusable.

Graphics Only Constructs: Nodes can be connected with lines to show topological information, but this is for visual guidance only. That is, one does not have to construct a path from Node A to Node B in order to route traffic from Node A to Node B. (Enforcing line connections can lead to a morass of lines in highly connected systems that provides more confusion than visual help.) In addition, there is a "virtual" node that can be used strictly for graphical presentation nothing is actually routed to / from it.

5.4 Using QNPET (MASE)

The view graphs contain details on using QNPET, and will be discussed in detail in class, followed by hands on lab sessions. Reference [6] also contains some examples of using QNPET (MASE) for data communications performance analysis.

Lab Exercise 5.1: Build the simple M/M/1 model discussed in the viewgraphs, i.e. with an exponential inter arrival time, mean 2 time units and an exponential service time, mean 1 time unit.

References

[1] Q+ Users Guide, Vol. I, AT&T Bell Labs, Holmdel, NJ.
[2] OPNET MODELER External Interface Manual, Release 2.4, MIL 3, inc., Washington, DC, 1994.
[3] SIMCRIPT II.5 Reference Handbook, CACI Products Company, La Jolla, CA, 1993.
[4] A. M. Law and M. G. McComas, "Simulation Software for Communications Networks: The State of the Arts," IEEE Communications Magazine, pp. 44 - 50, Vol. 32, No. 3, 1994.
[5] "F. E. Cellier, Q. Wang and B. P. Zeigler, "A Five Level I-Hierarchy for the Management of Simulation Models", Proceedings of the 1990 Winter Simulation Conference, Piscataway, NJ, IEEE Press, pp. 55-60, 1990.
[6] A. A. Fredericks and Wen Jing, "MASE - an Integrated Environment for Modeling, Analysis, Simulation and Engineering of Computer / Communications Systems", Proceeding of GLOBECOM '95.
[7] B. Melamed and R-J.T. Morris, " Visual Simulation: The Performance Analysis Workstation", IEEE Computer No. 18, 1985, pp. 87-94
[8] W. Whitt, "Open and Closed Models for Networks of Queues", AT&T BLTJ, Vol. 63, No. 9, Nov. 1984, pp. 1431.
[9] A. Fredericks, "An Approximation Method for Analyzing a Virtual Circuit Based LAN - Solving the Simultaneous Resource Possession Problem", Teletraffic Analysis and Computer Performance Evaluation, O.J. Boxma et al, ed., Elsevier Science Publishers B.V., (North Holland), 1986, pp. 63-74
[10] W. Jing and A-A. Fredericks, "An Open Object- Oriented Simulation System for Communications Networks", Proceeding of GLOBECOM '95

Performance Evaluation - Unit 5

The Queueing Network Performance Engineering Tool (QNPET - MASE)

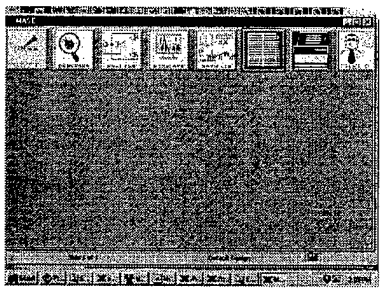
1

QNPET Toolset

- Builder / Editor
- Consistency Checker
- Analyzer
- Simulator
- Engineer's Assistant
- Browser
- Help

2

QNPET Toolbar



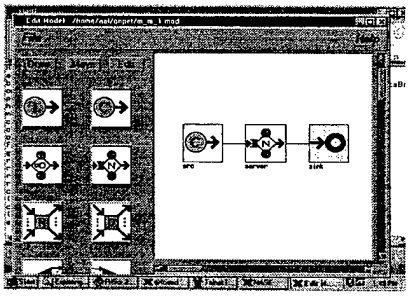
3

Builder / Editor

- Build / Save / Retrieve /Edit / Output Generalized Queueing Network Models
- Build with Icons / Parameterize with Forms
- Data Entry (Errors) Minimized with Use of Lists of Choices, Automatic Creation of Paired Nodes, etc.
- Built (Consistent) Models Runnable - No Compiling
- Supports "Note Making"

4

M/M/1 Model on Palette



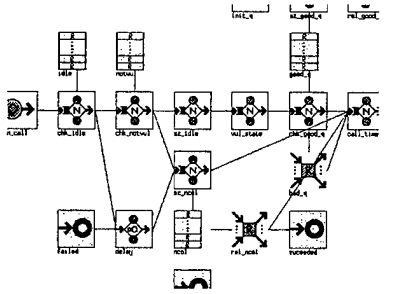
5

QNPET Modeling Constructs

- Sources (continuous and init) and Sink
- Service and Delay
- Resource Provider and "Splitter"
- Router and Distributor
- Split / Join (Fragment / Assemble, Fork / Synchronize, Replicate)
- Multiplex / Demultiplex
- Submodel and Submodel Template

6

QNPET Model of CSMA/CD With Environmental Effects



7

Consistency Checker

- Ensures Model Meaningful to Load
- Variety of Model Consistency Checks
- Analyzer Provides Additional Checks:
  - stability of network
  - consistency of resource seize / release process
  - consistency of split / join process

8

Analyzer

- Supplements Consistency Checker
- Exact Solution for Simple Open Networks
- Approximate Solutions for More Complex Systems
- Not All Networks Can Be Analyzed

9

Simulator

- Event / Delta / Continuous Run Modes
- Event / Delta and Final Statistics
- Pause with Parameter Changing
- ASCII Results Available Immediately
- Browser Can Be Used To Customize Results View

10

Engineer's Assistant

- Collection of Analytic Tools for Single Service Centers
- Engineering as well as Evaluation Capability
- Useful for Simulation Validation and Approximate Parameter Engineering

11

Other Tools

- Browser
  - browse results files
  - customize view
- Window Print
  - screen dump a window (e.g., model, results, etc.)
- Help

12

Executing QNPET

- On CSLAB
  - type "mase" at prompt
  - save models with .mod subscript
- Remote Login
  - set your display (e.g., in csh, "setenv DISPLAY machine (or ip address):0
  - continue as above

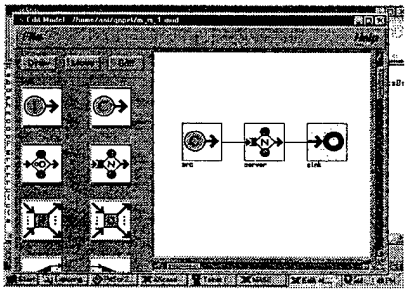
13

Modeling an M/M/1 Queue

- First Build an M/M/1 Queueing Model
  - exponential service times, mean = 1 (unit)
  - Poisson arrivals, rate .5 (mean interarrival time = 2 (units))
  - also need a sink to destroy transactions
  - save model as an mm1.mod (suffix important)
- Run the Consistency Checker on the Model
- Run the Simulation (Validate Results)

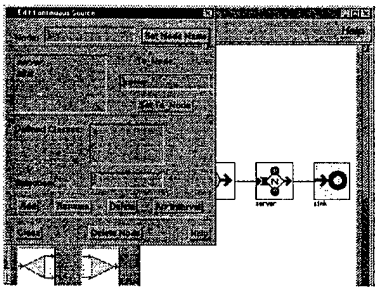
14

M/M/1 Model on Palette



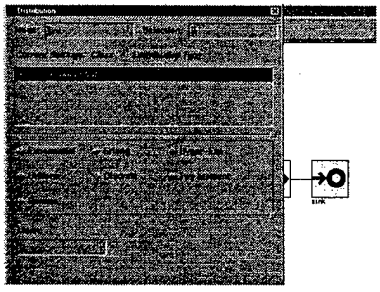
15

Main Source Editing Window



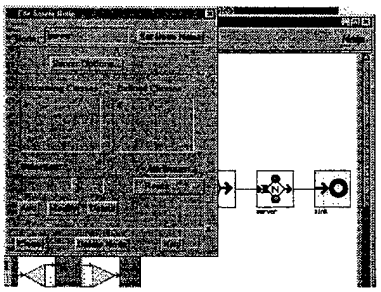
16

Interarrival Time Distribution at Source



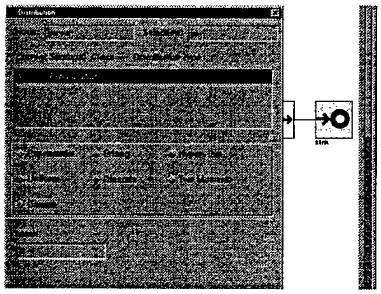
17

Main Server Editing Window



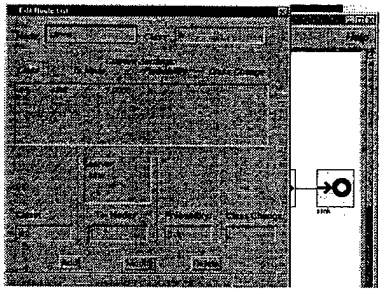
18

Server Service Time Distribution



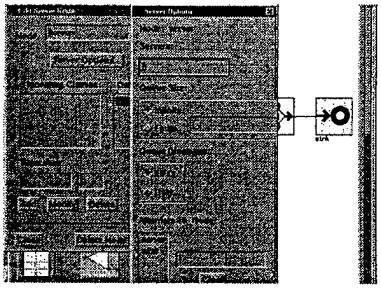
19

Routing for Server Node



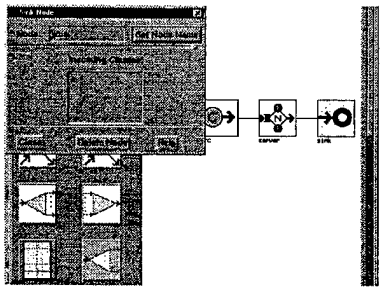
20

Server Options Editing Window



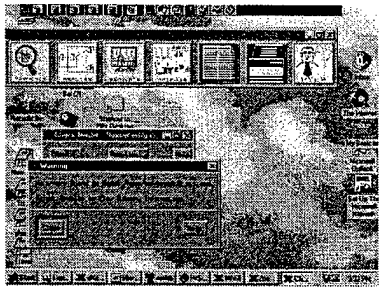
21

Main Sink Editing Window



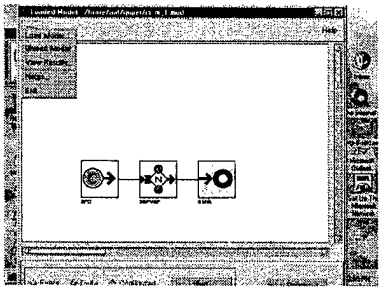
22

Consistency Checker Results



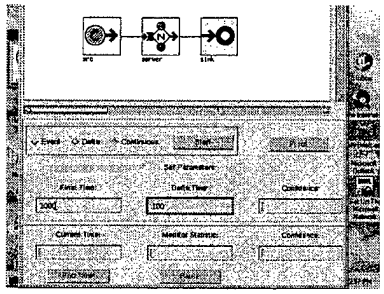
23

Simulation File Menu



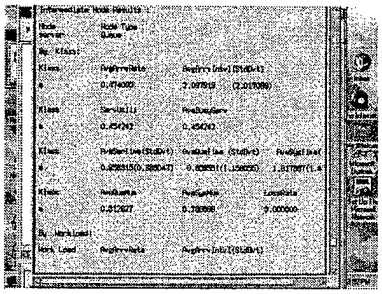
24

Instrumenting the Simulation



25

Result at End of 1000 units



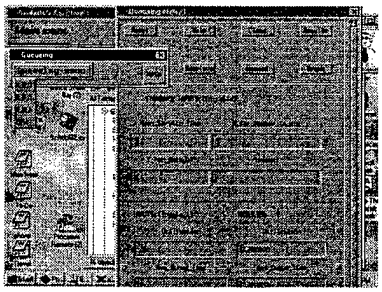
26

Validating Results

- For M/M/1 Queue, $\rho = \lambda X = .5x1 = .5$
- $T_s = X / (1 - \rho) = 2$ (units)
- Simulation After 1000 Units Shows $T_s = 1.62$
- What's the Problem?
- Try Running For 10000 Units

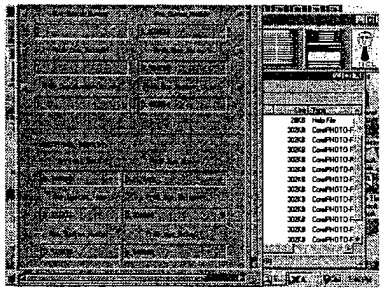
27

Analyst's Assistant M/M/1 Form



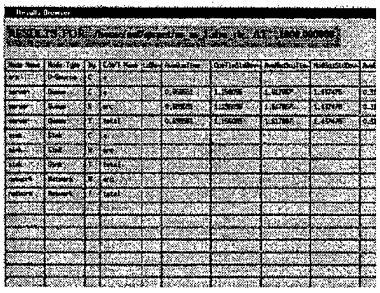
28

Additional Results for M/M/1



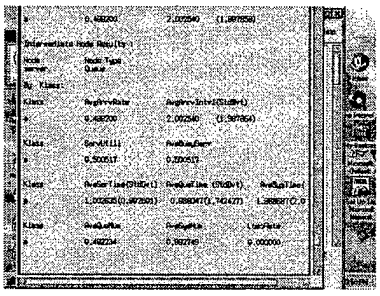
29

Viewing Results with the Results Browser



30

Results After 10000 Units



31

Appendix B

AMASE and the High Level Architecture

The AMASE modeling, analysis and simulation environment is essentially HLA compliant in that it either currently meets all HLA rules or can be readily enhanced to do so. The key HLA rules are divided into two categories: i) those pertaining to federations and ii) those pertaining to federates. Both sets of rules are pertinent to AMASE since it is both an environment for building federations and it contains a library of federates (currently under construction). We discuss these two sets of rules below and how they relate to AMASE. We then discuss the work planed under this task.

Rules for federations: For completeness we summarize first the five HLA rules for federations and then comment on AMASE compliance.

Rule 1: Federations shall have a HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).

AMASE Compliance: While we have not specifically used the OMT, AMASE is a highly structured environment (all of the tenants of object-oriented design have been followed) for building federations and as such it should take a minimal amount of effort to build the required documentation in the appropriate OMT format.

Rule 2: In a federation, all representation of objects in the FOM shall be in federates, not in the runtime infrastructure (RTI).

AMASE Compliance: The AMASE architecture in Figure 1 shows clearly the separation of concerns. AMASE's RTI is purposely made as lean as possible. The Central Control (CC) is essentially a message-processing agent. It receives messages from the Graphical User Interface (GUI) e.g. to build, initialize, run a simulation of a confederation. In additional to a global clock, it maintains two lists, a Pending Message list (PM list) and a Federate Event list (FE list). It sends / receives messages from the federates (and the GUI) coordinating the simulation timing. There are no simulation objects in the RTI.

Rule 3: During a federation execution, all exchange of FOM data among federates shall occur via the RTI.

AMASE Compliance: As shown in figure 1, all federates must communicate via the RTI. Indeed, since the federates are allowed to be in different communicating classes (i.e., exchange different message objects) the RTI must relay messages so that it can convert the format when appropriate.

Rule 4: During a federation execution, federates shall interact with the runtime infrastructure (RTI) in accordance with the HLA interface specifications.

AMASE Compliance: AMASE has a highly structured interface specification that all federates must comply with to ensure that they are plug compatible within AMASE. These specific (HLA) interfaces are the only means for federates to communicate with the RTI.

Rule 5: During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.

AMASE Compliance: If the developers of the federates adhere to this rule (all library federates in AMASE do) then the AMASE RTI will ensure that the federation does also.

Rules for federates

Rule 6: Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template (OMT).

AMASE Compliance: The specifications that AMASE provides for the addition of library simulation modules (federates) is highly structured and can readily be translated into the appropriate OMT format.

Rule 7: Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.

Rule 8: Federates shall be able to transfer and/or accept ownership of attributes dynamically during a federation execution, as specified in their SOM.

Rule 9: Federates shall be able to vary the conditions (e.g., thresholds) under which they provide updates of attributes of objects, as specified in their SOM.

AMASE Compliance: Rules 7 – 9 essentially provide for the capability of an orderly exchange of data, objects, etc, during runtime via the RTI facilities. Again, the interface specifications and parameterization of AMASE library modules is structured appropriately for supporting these rules.

Rule 10: Federates shall be able to manage local time in a way which will allow them to coordinate data exchange with other members of a federation.

AMASE Compliance: With AMASE the use of a Local Federate stub (LF stub – see figure 1) not only allows for remote execution of a federate, but also allows for a variety of simulation synchronization methods. The stub contains both the needed communications capabilities and the simulation control for the desired mode. From the main RTI's viewpoint, all federates reside locally and are running in a synchronous simulation mode. The stub logic can be configured to run in a synchronous simulation mode as well as in a bounded event time error mode (a new mode) or in rolback mode – if the federate it is a stub for support rollback.

Figure 1 AMASE Architecture